Episodic Reinforcement Learning by Logistic Reward-Weighted Regression

Daan Wierstra[1], Tom Schaul[1], Jan Peters[2], and Juergen Schmidhuber[1,3]

[1] IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
[2] MPI for Biological Cybernetics, Spemannstrasse 38, 72076 Tübingen, Germany
[3] Technical University Munich, D-85748 Garching, Germany

Abstract. It has been a long-standing goal in the adaptive control community to reduce the generically difficult, general reinforcement learning (RL) problem to simpler problems solvable by supervised learning. While this approach is today's standard for value function-based methods, fewer approaches are known that apply similar reductions to policy search methods. Recently, it has been shown that immediate RL problems can be solved by reward-weighted regression, and that the resulting algorithm is an expectation maximization (EM) algorithm with strong guarantees. In this paper, we extend this algorithm to the episodic case and show that it can be used in the context of LSTM recurrent neural networks (RNNs). The resulting RNN training algorithm is equivalent to a weighted self-modeling supervised learning technique. We focus on partially observable Markov decision problems (POMDPs) where it is essential that the policy is nonstationary in order to be optimal. We show that this new reward-weighted logistic regression used in conjunction with an RNN architecture can solve standard benchmark POMDPs with ease.

1 Introduction

In order to apply reinforcement learning (RL) to real-life scenarios it is often essential to deal with hidden and incomplete state information. While such problems have been discussed in the framework of partially observable Markov decision problems for a long time, this class of problems still lacks a satisfactory solution [1]. Most known methods to solve small POMDPs rely heavily on knowledge of the complete system, typically in the form of a belief-estimator or filter. Without such important information, the problem is considered intractable even for linear systems, and is not distinguishable from non-Markovian problems [2]. As a result, both POMDPs and non-Markovian problems largely defy traditional value function based approaches.

While policy search based approaches can be applied even with incomplete state information [3], they cannot yield an optimal solution unless the policy has an internal state [4]. As the internal state only needs to represent the features of the belief state and not all of its components, a function approximator with an internal state would be the ideal representation of a policy, and a recurrent

V. Kůrková et al. (Eds.): ICANN 2008, Part I, LNCS 5163, pp. 407–416, 2008.

neural network constitutes one of the few choices. It offers an internal state estimator as a natural component and is well-suited for unstructured domains.

However, the training of recurrent neural networks in the context of reinforcement learning is non-trivial as traditional methods often do not easily transfer to function approximators, and even if they do transfer, the resulting methods such as policy gradient algorithms do no longer employ the advantages of the strong results obtained for supervised learning. As a way out of this dilemma, we fall back onto a classical goal of reinforcement learning, i.e., we search for a way to reduce the reinforcement learning problem to a supervised learning problem where a multitude of methods exists for training recurrent neural networks. In order to do so, we re-evaluate the recent result in machine learning, that reinforcement learning can be reduced onto *reward-weighted regression* [5] which is a novel algorithm derived from Dayan & Hinton's [6] expectation maximization (EM) perspective on RL. We show that this approach generalizes from immediate rewards to episodic reinforcement learning to form Episodic Logistic Reward-Weighted Regression (ELRWR).

As a result, we obtain a novel, general learning method for memory-based policies such as recurrent neural networks in model-free partially observable environments, that is, a method that does not require prior knowledge of any of the dynamics of the problem setup. Using similar assumptions as in [5], we can show that episodic reinforcement learning can be solved as a utility-weighted nonlinear logistic regression problem in this context, which greatly accelerates the speed of learning. We obtain a reinforcement learning setup which is well-suited for training long short-term memory (LSTM) [7] recurrent neural networks, using the E-step of the algorithm to generate weightings for training the memory-capable LSTM network in the M-step. Intuitively, the network is trained to imitate or *self-model* its own actions, but with more successful episodes weighted more heavily than the unsuccessful ones, resulting in a convergence to an ever better policy. We evaluate ELRWR on a number of standard POMDP benchmarks, and show that this method provides a viable alternative to more traditional RL approaches.

2 Preliminaries

In this section, we will state our general problem, define our notation and briefly discuss long short-term memory (LSTM) recurrent neural networks.

2.1 Reinforcement Learning – Generalized Problem Statement

First, let us introduce the RL framework used in this paper and the corresponding notation. The environment produces a state g_t at every time step. Transitions from state to state are governed by a probability function $p(g_{t+1}|a_{1:t}, g_{1:t})$ unknown to the agent but dependent upon all previous actions $a_{1:t}$ executed by the agent and all previous states $g_{1:t}$ of the system. Let r_t be the reward assigned to the agent at time t, and let o_t be the corresponding observation produced by the

environment. We assume that both quantities are governed by fixed distributions $p(o|g)$ and $p(r|g)$, solely dependent on state g.

In the more general reinforcement learning setting, we require that the agent has a memory of the generated experience consisting of finite episodes. Such episodes are generated by the agent's operations on the (possibly stochastic) environment, executing action a_t at every time step t, after observing o_t which depends solely on g_t. Observation o_t includes special 'observation' r_t (the reward). We define the *observed history*[1] h_t as the string or vector of observations and actions up to moment t since the beginning of the episode: $h_t = \langle o_1, a_1, o_2, a_2, \ldots, o_{t-1}, a_{t-1}, o_t \rangle$. The complete history H has finite length $T(H)$, and includes the unobserved states and is given by $H_T = \langle h_T, g_{1:T} \rangle$. At any time t, the statistic $R_t = (1-\gamma) \sum_{k=t}^{T(H)} r_k \gamma^{t-k-1}$ denotes the *return* at time t where $0 < \gamma < 1$ denotes a discount factor.

The expectation of this return R_t at time $t = 1$ is also the measure of quality of our policy and, thus, the objective of reinforcement learning is to determine a policy which is optimal with respect to the expected future discounted rewards or expected return

$$J = E[R_1] = (1-\gamma)\mathbf{E}\left[\sum_{t=1}^{T} \gamma^t r_t\right]. \tag{1}$$

An optimal or near-optimal policy in a non-Markovian or partially observable Markovian environment requires that the action a_t is taken depending on the *entire* preceding history. However, in most cases, we will not *need* to store the whole string of events but only sufficient statistics $S(h_t)$ of the events which we call the internal memory of the agent's past. Thus, a stochastic policy π can be defined as $\pi(a|h_t) = p(a|S(h_t);\theta)$, implemented as a recurrent neural network (RNN) with weights θ and stochastically interpretable output neurons implemented as a softmax layer. This produces a probability distribution over actions, from which actions a_t are drawn $a_t \sim \pi(a|h_t)$.

2.2 LSTM Recurrent Neural Networks

RNNs have attracted some attention in the past decade because of their simplicity and potential power. However, though powerful in theory, they turn out to be quite limited in practice due to their inability to capture long-term time dependencies – they suffer from the problem of *vanishing gradient* [8], the fact that the gradient signal vanishes as the error signal is propagated back through time. Because of this, events more than 10 time steps apart can typically not be related.

One method purposely designed to avoid this problem is long short-term memory (LSTM) [7], which constitutes a special RNN architecture capable of capturing long term time dependencies. The defining feature of this architecture is that it consists of a number of *memory cells*, which can be used to store activations

[1] Note that such histories are also called path or trajectory in the literature.

for an arbitrarily long time. Access to the memory cell is *gated* by units that learn to open or close depending on the context.

LSTM networks have been shown to outperform other RNNs on numerous time series requiring the use of deep memory [9]. Therefore, they seem well-suited for usage in POMDP algorithms for complex tasks requiring deep memory. In the context of reinforcement learning, RNNs are usually used to *predict value*, however, we use them to *control* an agent directly, to represent a controller's policy which receives observations and produces action probabilities at every time step.

Our LSTM networks are trained using backpropagation through time (BPTT) [10], where sequences of ⟨input, target, weighting⟩ samples are used to train the network to minimize a (weighted) error function.

3 Logistic Reward-Weighted Regression for Recurrent Neural Networks

Intuitively, it is clear that the general reinforcement learning problem is related to supervised learning problems as the policy should *match* previously taken motor commands such that episodes are more likely to be reproduced if they had a higher return. The network is trained to imitate or *self-model* its own actions, but with more successful episodes weighted more heavily than the unsuccessful ones, resulting in a convergence to an ever better policy. In this section, we will solidify this approach based on [5], and extend the previous results from a single-step, immediate reward scenario to the general episodic case.

We first discuss our basic assumptions and introduce reward-shaping. Subsequently, we show how a utility-weighted mean-squared error emerges from the general assumptions for an expectation maximization algorithm. Finally, we present the entire resulting algorithm.

3.1 Optimizing Utility-Transformed Returns

Let the return $R(H)$ be some measure of the total reward accrued during a history (e.g., $R(H)$ could be the average of the rewards for the average reward case or the future discounted sum for the discounted case), and let $p(H|\theta)$ be the probability of a history given policy-defining weights θ, then the quantity the algorithm should be optimizing is the expected return

$$J = \int_H p(H|\theta)R(H)dH. \tag{2}$$

This, in essence, indicates the expected return over all possible histories, weighted by their probabilities under policy π.

While a goal function such as found in Eq. (2) is sufficient in theory, algorithms which plainly optimize it have major disadvantages. They might be too aggressive when little experience is available, and converge prematurely to the best solution they have seen so far. On the opposite extreme, they might prove

to be too passive and be biased by less fortunate experiences. Trading off such problems has been a long-standing challenge in reinforcement learning. However, in decision theory, such problems are surprisingly well-understood [11]. In that framework it is common to introduce a so-called utility transformation $u_\tau(R)$ which has to fulfill the requirement that it scales monotonically with R, is semi-positive and integrates to a constant. Once a utility transformation is inserted, we obtain an expected utility function given by

$$J_u(\theta) = \int p(H|\theta) u_\tau(R(H))\, dH. \tag{3}$$

The utility function $u_\tau(R)$ is an adjustment for the aggressiveness of the decision making algorithms, e.g., if it is concave, it's attitude is risk-averse while if it is convex, it will be more likely to consider a reward more than a coincidence. Obviously, it is of essential importance that this risk function is not manually tweaked but rather chosen such that its parameters τ can be controlled adaptively in accordance with the learning algorithm.

In this paper, we will consider one simple utility transformation function, the soft-transform $u_\tau(r) = \tau \exp(\tau r)$ also used in [5].

3.2 Expectation Maximization for Reinforcement Learning

Analogously as in [5,6], we can establish the lower bound

$$\log J_u(\theta) = \log \int q(H) \frac{p(H|\theta) u_\tau(R(H))}{q(H)} dH \tag{4}$$

$$\geq \int q(H) \log \frac{p(H|\theta) u_\tau(R(H))}{q(H)} dH \tag{5}$$

$$= \int q(H) \left[\log p(H|\theta) + \log u_\tau(R(H)) - \log q(H)\right] dH \tag{6}$$

$$= \mathcal{F}(q, \theta, \tau), \tag{7}$$

due to Jensen's inequality with the additional constraint $0 = \int q(H) dH - 1$. This points us to the following EM algorithm:

Proposition 1. *An Expectation Maximization algorithm for optimizing both the expected utility as well as the reward-shaping is given by*

$$\textit{E-Step:}\quad q_{k+1}(H) = \frac{p(H|\theta) u_\tau(R(H))}{\int p(\tilde{H}|\theta) u_\tau\left(R(\tilde{H})\right) d\tilde{H}}, \tag{8}$$

$$\textit{M-Step Policy:}\quad \theta_{k+1} = \arg\max_\theta \int q_{k+1}(H) \log p(H|\theta) dH, \tag{9}$$

$$\textit{M-Step Utility Adaptation:}\quad \tau_{k+1} = \arg\max_\tau \int q_{k+1}(H) \log u_\tau(R(H))\, dH. \tag{10}$$

Proof. The E-Step is given by $q = \text{argmax}_q \mathcal{F}(q, \theta, \tau)$ while fulfilling the constraint $0 = \int q(H)dH - 1$. Thus, we have a Lagrangian $L(\lambda, q) = \mathcal{F}(q, \theta, \tau) - \lambda$. When differentiating $L(\lambda, q)$ with respect to q and setting the derivative to zero, we obtain $q^*(H) = p(H|\theta)u_\tau(R(H))\exp(\lambda - 1)$. We insert this back into the Lagrangian obtaining the dual function $L(\lambda, q^*) = \int q^*(H)dH - \lambda$. Thus, by setting $dL(\lambda, q^*)/d\lambda = 0$, we obtain $\lambda = 1 - \log \int p(H|\theta)u_\tau(R(H))\, dH$, and solving for q^* implies Eq (8). The M-steps compute $[\theta_{k+1}, \tau_{k+1}]^T = \text{argmax}_{\theta,\tau} \mathcal{F}(q_{k+1}, \theta, \tau)$. We can maximize $\mathcal{F}(q_{k+1}, \theta, \tau)$ for θ, τ independently, which yields Eqs. (9,10).

3.3 The Utility-Weighted Error Function for the Episodic Case

For every reinforcement learning problem, we need to establish the cost function $\mathcal{F}(q_{k+1}, \theta, \tau)$ and maximize it in order to derive an algorithm. For episodic reinforcement learning, we first need to recap the general settings. We can denote the probabilities $p(H|\theta)$ of histories H by

$$p(H|\theta) = p(\langle o_1, g_1 \rangle) \prod_{t=2}^{T(H)} p(\langle o_t, g_t \rangle | h_{t-1}, a_{t-1}, g_{1:t-1}) \pi(a_{t-1}|h_{t-1}) \tag{11}$$

which are dependent on an unknown initial state and observation distribution $p(\langle o_1, g_1 \rangle)$, and on unknown state transition function $p(g_{t+1}|a_{1:t}, g_{1:t})$. However, the policy $\pi(a_t|h_t)$ with parameters θ is known, where h_t denotes the history which is collapsed into the hidden state of the network.

It is clear that the expectation step has to follow by simply replacing the expectations by sample averages. Thus, we have

$$q_{k+1}(H_i) = \frac{u_\tau(R(H_i))}{\sum_{j=1}^{N} u_\tau(R(H_j))} \tag{12}$$

as E-step. We define $U_N = \sum_{j=1}^{N} u_\tau(R(H_j))$ as summed utility of all N histories.

The maximization or M-step of the algorithm requires optimizing θ such that $\mathcal{F}(q_{k+1}, \theta, \tau)$ is maximized. In order to optimize θ we realize that the probability of a particular history is simply the product of all actions and observations given subhistories (Eq. 11). Taking the log of this expression transforms this large product into a sum

$$\log p(H|\theta) = (\text{const}) + \sum_{t=1}^{T(H)} \log \pi(a_t|h_t) \tag{13}$$

where most parts are not affected by policy-defining parameters θ, i.e., are constant, since they are solely determined by the environment. Thus, when optimizing θ we can ignore this constant and purely focus on the outputs of the policy, optimizing the expression

$$\mathcal{F}(q_{k+1}, \theta, \tau) \propto \sum_{i=1}^{N} q_{k+1}(H_i) \log p(H_i|\theta) = \sum_{i=1}^{N} \frac{u_\tau(R(H_i))}{U_N} \sum_{t=1}^{T(i)} \log \pi(a_t^i|h_t^i), \tag{14}$$

where a_t^i denotes an action from the complete history i at time t and h_t^i denotes the collapsed history i up to time-step t.

3.4 Logistic Reward-Weighted Regression for LSTMs

As we are interested in using recurrent neural networks as policies while avoiding the vanishing gradient problem, it is a logical choice that our policy $\pi(a_t|h_t)$ with parameters θ will be represented by a long short-term memory (LSTM) recurrent neural network. Here, we still condition on the the history h_t of our sequence up to time step t as it is collapsed into the hidden state of the network. We use a standard LSTM architecture where the discrete actions are drawn from a softmax output layer, that is, we have

$$\pi(a_t|h_t) = \frac{\exp(f(a_t, h_t))}{\sum_{a=1}^{A} \exp(f(a, h_t))} \tag{15}$$

for all A actions where the output of the neurons are $f(a_t, h_t)$. We can compute the cost function $\mathcal{F}(q_{k+1}, \theta, \tau)$ for this policy and obtain the utility-weighted conditional likelihood function

$$\mathcal{F}(q_{k+1}, \theta, \tau) = \sum_{i=1}^{N} \frac{u_\tau(R(H_i))}{U_N} \sum_{t=1}^{T(i)} \left(a_t^i f(a_t^i, h_t^i) - \log \sum_{a=1}^{A} \exp(f(a, h_t)) \right). \tag{16}$$

This optimization problem is equivalent to a weighted version of logistic regression [12]. As $f(a, h_t)$ is linear in the parameters of the output layer, these can be optimized directly. The hidden state related parameters of $f(a, h_t)$ can be optimized using backpropagation through time (BPTT) of the LSTM architecture. Both linear and nonlinear logistic regression problems cannot be solved in one single shot. Nevertheless, it is straightforward to show that the second-order expansion simply yields a linear regression problem which is equivalent to a Newton-Rapheson step on the utility-weighted conditional likelihood. As a result, we have an approximate regression problem

$$\mathcal{F}(q_{k+1}, \theta, \tau) \approx \sum_{i=1}^{N} \frac{u_\tau(R(H_i))}{U_N} \sum_{t=1}^{T(i)} \left(a_t^i - \pi(a_t^i|h_t^i) \right)^2, \tag{17}$$

which is exactly the utility-weighted squared error. Optimizing this expression by gradient descent allows us to use standard methods for determining the optimum. Nevertheless, there is a large difference in comparison to regular reward-weighted regression where the regression step can only be performed once – instead we can perform multiple BPTT training steps until convergence. In order to prevent overfitting we use the common technique of early stopping, assigning the sample histories to two separate batches for training and validation. While this supervised training scheme requires a relatively large demand in computation per sample history, it also reduces the number of episodes necessary for the policy to converge. Lastly, the update of τ optimizing Eq. 8 follows [5] as $\tau_{k+1} = \frac{\sum_{i=1}^{N} u_\tau(R(H_i))}{\sum_{i=1}^{N} u_\tau(R(H_i))R(H_i)}$ The complete algorithm pseudocode is displayed in Algorithm 1.

Algorithm 1. Episodic Logistic Reward-Weighted Regression

Initialize θ, training batch size N, $\tau = 1$, $k = 1$.
repeat
  for $i = 1 \dots N$ **do**
    Sample episode $H_i = \langle o_1, a_1, o_2, a_2, \dots, o_{T-1}, a_{T-1}, o_T \rangle$ using policy π.
    Evaluate return for $t = 1 : R(H_i)$.
    Compute utility of H_i as $u_\tau(R(H_i))$.
  end for
  Train weights θ of policy π until convergence with BPTT to minimize
  $$\mathcal{F}(q_{k+1}, \theta, \tau) \approx \sum_{i=1}^{N} \frac{u_\tau(R(H_i))}{U_N} \sum_{t=1}^{T(i)} \left(a_t^i - \pi(a_t^i | h_t^i) \right)^2,$$
  using validation sample histories for early stopping.
  Recompute $\tau \leftarrow \frac{\sum_{i=1}^{N} u_\tau(R(H_i))}{\sum_{i=1}^{N} u_\tau(R(H_i)) R(H_i)}$.
  $k \leftarrow k + 1$
until stopping criterion is met

4 Experiments

We experimented on 5 small POMDP benchmarks commonly used in the literature. The CheeseMaze, Tiger problem, Shuttle Docking benchmark and the 4x3Maze [13,14] are all classic POMDP problems which range from 2 to 11 states, with 2 to 7 observations. The last experiment was the T-maze [15], which was designed to test an RL algorithm's ability to correlate events far apart in history. It involves having to *learn* to remember the observation from the first time step until the episode ends. Its difficulty, depending on corridor lengths, can be adjusted. We investigated the T-Maze with corridor lengths 3, 5 and 7.

The policy was represented as an LSTM network, with input layer size dependent on the observation dimension, a hidden layer containing 2 LSTM memory cells, and a softmax output layer with size dependent on the number of actions applicable to the environment. The only tunable parameter, batch size N, was always set to 30, except for the CheeseMaze and the T-Maze, where it was set to 75. It was found that the algorithm is very robust to the particular setting of this parameter. One third of all batch sample episodes was used for validation in our early stopping scheme to prevent overfitting.

The specific settings for weighted supervised learning are of minor importance (assuming that the number of episodes determines performance), since we train every batch until (early stopping) convergence. Concretely, the LSTM network was initialized with weights uniformly distributed between -0.1 and 0.1. It was trained with BPTT using learning rate 0.002 and momentum 0.95, while weightings were used that are proportional to the self-adapting soft-transform $u_\tau(r) = \tau \exp(\tau r)$, but normalized such that the maximal weighting in every batch was always 1. All experiments consisted of 100 consecutive EM-steps, and were repeated 25 times to gain sufficient statistics.

Table 1. This table shows results averaged over 25 runs. Displayed are the average rewards and std. deviations obtained for the trained policy (Normal) after 100 EM steps, its greedy variant (Greedy) which always takes the learned policy's most likely action, the optimal policy manually calculated for each problem (Optimal), and a randomized policy (Random) as a reference. Shown results include statistics for T-Mazes with corridor lengths 3, 5 and 7.

Policy	Optimal	Random	Normal	Greedy
Tiger	6.7	-36	-6.5 ± 4.9	-5.7 ± 7.4
ShuttleDocking	1.69	-.31	$.709 \pm .059$	0.0 ± 0.0
4x3Maze	.27	.056	$.240 \pm .085$	$.246 \pm .092$
CheeseMaze	.257	.072	$.177 \pm .032$	$.212 \pm .057$
T-Maze3	1.0	.166	$.917 \pm .043$	1.0 ± 0.0
T-Maze5	0.666	.046	$.615 \pm .032$	$.662 \pm .021$
T-Maze7	0.5	.002	$.463 \pm .008$	$.484 \pm .090$

The results are shown in Table 1, which includes both the results for a random policy and the manually calculated optimal policy for each task as a reference. We can see that all problems converged quickly to a good solution, except for the Shuttle Docking benchmark where 13 out of 25 runs failed to converge to an acceptable solution. This might be due to the problem's inherently stochastic nature, which possibly induces the algorithm to converge prematurely. The T-Maze results are significantly less impressive than found in [15] and [4], where corridor lengths of 70 and 90 are reached. However, the result of solving T-Maze length 7 in less than 100 EM steps with batch size 75 constitutes a competitive result.

Good results were obtained without any fine tuning. This encourages us to expect that extensions of the approach will produce a rather general POMDP solver. Such extensions could include the properly re-weighted reuse of information from previous batches, resetting network weights for every EM step, and various improvements to the supervised learning scheme. Future research will include the investigation of the possibility of the use of value-functions and the time-specific reward-attributions to alleviate the credit assignment problem, by shifting responsibilities from entire sequences to single actions.

5 Conclusion

In this paper we introduced a novel, surprisingly simple EM-derived episodic reinforcement learning algorithm that learns from temporally delayed rewards. The method can learn to deal with partially observable environments by using long short-term memory, the parameters of which are updated using utility-weighted logistic regression as training method. The successful application of this algorithm to a number of POMDP benchmarks shows that reward-weighted regression is a promising approach for episodic reinforcement learning, even in non-Markovian settings.

Acknowledgments

This research was funded by SNF grants 200021-111968/1 and 200021-113364/1.

References

1. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101 (1998)
2. Aoki, M.: Optimization of Stochastic Systems. Academic Press, New York (1967)
3. Baxter, J., Bartlett, P.: Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research 15, 319–350 (2001)
4. Wierstra, D., Foerster, A., Peters, J., Schmidhuber, J.: Solving deep memory pomdps with recurrent policy gradients. In: de Sá, J.M., Alexandre, L.A., Duch, W., Mandic, D.P. (eds.) ICANN 2007. LNCS, vol. 4668, pp. 697–706. Springer, Heidelberg (2007)
5. Peters, J., Schaal, S.: Reinforcement learning by reward-weighted regression for operational space control. In: Proceedings of the International Conference on Machine Learning (ICML) (2007)
6. Dayan, P., Hinton, G.E.: Using expectation-maximization for reinforcement learning. Neural Computation 9(2), 271–278 (1997)
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
8. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kremer, S.C., Kolen, J.F. (eds.) A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, Los Alamitos (2001)
9. Schmidhuber, J.: RNN overview (2004), `http://www.idsia.ch/~juergen/rnn.html`
10. Werbos, P.: Back propagation through time: What it does and how to do it. Proceedings of the IEEE 78, 1550–1560 (1990)
11. Chernoff, H., Moses, L.E.: Elementary Decision Theory. Dover Publications (1987)
12. Kleinbaum, D.G., Klein, M., Pryor, E.R.: Logistic Regression, 2nd edn. Springer, Heidelberg (2002)
13. James, M.R., Singh, S., Littman, M.L.: Planning with predictive state representations. In: Proceedings 2004 International Conference on Machine Learning and Applications, pp. 304–311 (2004)
14. Bowling, M., McCracken, P., James, M., Neufeld, J., Wilkinson, D.: Learning predictive state representations using non-blind policies. In: ICML 2006: Proceedings of the 23rd international conference on Machine learning, pp. 129–136. ACM, New York (2006)
15. Bakker, B.: Reinforcement learning with long short-term memory. In: Advances in Neural Information Processing Syst., vol. 14 (2002)